A product-form solution for on-off components in PEPA

Nigel Thomas

School of Computing Science, Newcastle University, UK.
nigel.thomas@ncl.ac.uk

Abstract. This paper addresses the issue of how the behaviour of the model may be used to directly show product form results in general models without relying on additional insight from the modeller. To do this properties of the model are defined which allow the identification of model decompositions which are then shown to exhibit product by employing the reversed process.

1 Introduction

Throughout the history of analysing the performance of computer and communication networks, researchers have endeavoured to find efficient mechanisms for tackling models with extremely large state spaces. Within stochastic process algebra there are a number of approaches that have been investigated, including decomposition [1, 2], mean value analysis [3], and fluid approximations [4, 5].

One of the most powerful and mathematically attractive decomposition techniques are so-called product form solutions. Such solutions are derived on the basis that the components in the model are statistically independent in their steady state behaviour and so the steady state solutions for components may be found in isolation without the need to generate the entire state space of the model. Two general approaches have been used to identify product form solutions in stochastic process algebra, structural decomposition [6, 7] relying on identifying patterns in the model specification and reverse processes [9–11].

The aim of this paper is to address the issue of how the behaviour of the model may be used to directly show product form results in general models without relying on additional insight from the modeller. To do this properties of the model are defined which allow the components to be decoupled so that their marginal steady state distributions can be derived. Under certain conditions, these decomposed models give rise to a product form solution.

The paper begins by re-introducing Hillston's Markovian process algebra, PEPA [12], together with the set of concepts required to describe features of a model and then briefly discusses the notion of behavioural independence. In Section 3 the exploitation of behavioural independence is made in relation to simple product form decomposition. Finally some conclusions and future work directions are presented.

2 PEPA

A formal presentation of PEPA is given in [12], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that are negative exponentially distributed at given rates. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α, r) is described by the *type* of the activity, α, and the *rate* of the associated negative exponential distribution, r. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

$$P ::= (\alpha, r).P \mid P + Q \mid P/L \mid P \underset{L}{\bowtie} Q \mid A$$

- The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves like P. The component $P + Q$ behaves either like P or like Q, the resultant behaviour being given by the first activity to complete.
- The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ.
- Concurrent components can be synchronised, $P \underset{L}{\bowtie} Q$, such that activities in the *cooperation set* L involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to the activities of that type. The parallel combinator, $\|$, is used as shorthand to denote synchronisation with no shared activities, i.e. $P \| Q \equiv P \underset{\emptyset}{\bowtie} Q$.
- $A \overset{def}{=} P$ gives the constant A the behaviour of the component P.

In the following sections a number of properties of PEPA models will be referred to. Since the definitions of these properties are presented in detail elsewhere [12], only a general description of them is included here. Informally, it can be said that, two PEPA expressions are *isomorphic* if they give rise to Markov chains which are equivalent such that for every state in each Markov chain there is a corresponding state in the other with the same one-step transition rates to states which are similarly equivalent. A *derivative* is the "state" of a component defining its current behaviour. For example, if $P \overset{def}{=} (\alpha, r).Q$ then Q is a derivative of P and if $Q \overset{def}{=} (\beta, r_2).R$ then R is a derivative of both Q and P, and so on. The *derivative set*, $ds(P)$, is the set of all the possible derivatives of a component, P. The *current action type set* of a component P, $\mathcal{A}(P)$, contains all the action types (but not the rates) that are enabled in the current derivative of the component P. The *complete action type set* of a component P, $\boldsymbol{\mathcal{A}}(P)$, contains all the action types (but not the rates) that are enabled in any of the derivatives $P' \in ds(P)$, hence $\boldsymbol{\mathcal{A}}(P) = \bigcup_{P' \in ds(P)} \mathcal{A}(P')$. The *current action set* of a component P, $\mathcal{A}ct(P)$, contains all the actions (type and rate pairs) that are enabled in the current derivative of the component P. In addition it is necessary to construct a number of additional definitions.

Fertile action An action γ is said to be fertile in derivative P_i if $P_i \xrightarrow{\gamma} P_j$ and $i \neq j$.

Current fertile action type set The current fertile action type set of a component P, denoted $\mathcal{A}_f(P)$, is the set of all action types of actions that are fertile in the current derivative of P.

Complete fertile action type set The complete fertile action type set of a component P, denoted $\boldsymbol{\mathcal{A}_f}(P)$, is the set of all action types of actions that are fertile in at least one derivative of P.

3 Behavioural Independence

Put simply the notion of *behavioural independence* is simply that a component in a model behaves identically regardless of the current behaviour of the other components in the model. This property can be defined more formally thus:

The component P is said to be **behaviourally independent** in the model $P \underset{L}{\bowtie} Q$ if for every $P_i \in ds(P)$

$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_j)/\{\mathcal{A}_f(P_i) \cap L\}\}\right) =$$

$$\mathcal{A}ct\left((P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}(P_i \underset{L}{\bowtie} Q_k)/\{\mathcal{A}_f(P_i) \cap L\}\}\right)$$

$\forall\ Q_j, Q_k \in ds(Q)\ s.t.\ (P_i \underset{L}{\bowtie} Q_j), (P_i \underset{L}{\bowtie} Q_k) \in ds(P \underset{L}{\bowtie} Q)$

Obviously the trivial case for behavioural independence is where there are no shared actions, i.e. $P||Q$, however this is not the only case where components may be considered to be behaviourally independent. Furthermore, the fact that no actions are shared between two components does not mean they will always be behaviourally independent in the presence of other components. For example, in $(P||Q) \underset{L}{\bowtie} R$ the interaction between P and R may influence the interaction between Q and R, causing P and Q to be behaviourally *dependent*.

If a component is not behaviourally independent then it must be dependent on some other component to perform one of more actions during its evolution. This dependence is referred to by saying that component P *controls* component Q over action $K \subset L$ in $P \underset{L}{\bowtie} Q$ if the rate at which an action of type $k \in K$ can happen in $Q_i \in dsQ$ depends on the current derivative of P. Clearly, if P controls Q over K then Q cannot be behaviourally independent, but the independence, or otherwise, of P is not known by this statement.

The definition for behavioural independence given above relies on knowing the combination of each component's derivatives in every state of the underlying Markov chain; in essence, the entire state space would have to be explored. If the potential benefit of compositionality is to be realised then it is necessary that behavioural independence can be identified at the component level, without knowledge of the underlying CTMC. A number of simple specific cases in which P can be observed to be behaviourally independent in $P \underset{L}{\bowtie} Q$ can be stated.

- $L \cap \boldsymbol{\mathcal{A}_f}(P) = \emptyset$.

- Q is redundant in $P \underset{L}{\bowtie} Q$ and P is sequential.
- If an action $l \in (L \cap \boldsymbol{\mathcal{A}_f}(P))$ is enabled in $Q_i \in ds(Q)$ at rate α then it is also enabled in every $Q_j \in ds(Q)$ at rate α (α may be unspecified).
- If an action $l \in (L \cap \boldsymbol{\mathcal{A}_f}(P))$ is enabled in $Q_i \in ds(Q)$ at a rate $\alpha \geq Mr(l, P)$ then it is also enabled in every $Q_j \in ds(Q)$ at rate $\alpha_j \geq Mr(l, P)$ (note: α or α_j may be unspecified).

Note that the final condition is a numerical one whereas the others are structural only. In general numerical values are not considered at the model specification stage and so this last condition is of only limited use.

More formally, if one of the following conditions holds true $\forall \gamma \in \mathcal{A}(Q_i)$ and $\forall Q_i \in ds(Q)$ then P is behaviourally independent in $P \underset{L}{\bowtie} Q$.

1. if $\gamma \in \mathcal{A}_f(Q_i)$ such that $Q_i \xrightarrow{\gamma} Q_j$, $(i \neq j)$ then $\mathcal{A}ct(Q_i/\{\{\mathcal{A}(Q_i)/\{L \cap \boldsymbol{\mathcal{A}_f}(P)\}\}) = \mathcal{A}ct(Q_j/\{\mathcal{A}(Q_j)/\{L \cap \boldsymbol{\mathcal{A}_f}(P)\}\})$
2. if $\gamma \in \{\mathcal{A}_f(P_i) \cap L\}$ such that $P_i||Q_k \xrightarrow{\gamma} P_j||Q_l$, $(i \neq j)$ then

$$\mathcal{A}ct(Q_k/\{\mathcal{A}(Q_k)/\{L \cap \mathcal{A}_f(P_j) \cap \mathcal{A}_f(P_i)\}\}) =$$
$$\mathcal{A}ct(Q_l/\{\mathcal{A}(Q_l)/\{L \cap \mathcal{A}_f(P_j) \cap \mathcal{A}_f(P_i)\}\})$$

The conditions set out here can be used to test a model for behavioural independence at the component level, however they are unnecessarily strong, thus it is not possible to say that a component failing to meet these criteria is dependent in some way on another component.

3.1 Product form over components

As well as being used to identify independent behaviour leading to decomposition, behavioural independence and control can also be used to identify cases where product form solutions exist. Such a case is the queueing model with breakdowns illustrated below.

$$\begin{aligned} Queue_0 &\overset{def}{=} (arrival, \top).Queue_1 \\ Queue_i &\overset{def}{=} (arrival, \top).Queue_{i+1} + (service, \top).Queue_{i-1} \; 1 \leq j \leq N-1 \\ Queue_N &\overset{def}{=} (service, \top).Queue_{N-1} \end{aligned}$$

$$\begin{aligned} Server_{on} &\overset{def}{=} (fail, \xi).Server_{off} + (arrival, \lambda).Server_{on} + (service, \mu).Server_{on} \\ Server_{off} &\overset{def}{=} (repair, \eta).Server_{on} \end{aligned}$$

$$Queue_0 \underset{\{service, arrival\}}{\bowtie} Server_{on}$$

It is clear that the *Server* component is behaviourally independent in this model as neither of the shared actions affects its evolution. Similarly it is clear that the *Server* component controls the *Queue* component over the actions *service* and *arrival*. A number of other important factors are also apparent:

– All the actions of *Queue* are shared actions.
– All shared actions are enabled in $Server_{on}$.
– No shared actions are enabled in $Server_{off}$.
– The activity *fail* does not alter the current derivative of *Queue*.

These six factors mean that a product form solution exists over the *Server* and *Queue* components such that the joint steady state probabilities are given as:

$$p(Server_j|Queue_i) = p(Server_j).p(Queue_i)$$

where $j \in \{on, off\}$ and $0 \leq i \leq N$.

The model illustrated in here is reversible and it is possible to derive a product form solution using the characterisation derived in [8]. That approach required the identification of reversible components and the application of restrictions on the cooperation between them. This requires a detailed study of both the components and the interface, whereas the approach described here only requires a simple inspection of the components, only adherence to the five criteria.

1. One component, A, of a pair $A \underset{L}{\bowtie} B$ is behaviourally independent.
2. The other component, B, is controlled by A over all the actions in the cooperation set, L.
3. The complete action type set of A, $\boldsymbol{\mathcal{A}}(B)$ is contained within its interface, $\boldsymbol{\mathcal{A}}(B) = L$.
4. All actions in the cooperation set, L, are enabled in exactly one derivative of A.
5. No actions in the cooperation set, L, are enabled in any other derivative of A.

As long as these 5 stated conditions are not broken then the model illustrated above can be easily adapted to incorporate additional (non-reversible) features, such as batch service, without compromising the product form solution. Such a model is illustrated below.

$$\begin{aligned}
Queue_0 &\overset{def}{=} (arrival, \top).Queue_1 \\
Queue_i &\overset{def}{=} (arrival, \top).Queue_{i+1} + (service, \top).Queue_0 \; 1 \leq j \leq N-1 \\
Queue_N &\overset{def}{=} (service, \top).Queue_0
\end{aligned}$$

$$\begin{aligned}
Server_{on} &\overset{def}{=} (fail, \xi).Server_{off} + (arrival, \lambda).Server_{on} \\
&\quad +(service, \mu).Server_{on} \\
Server_{off} &\overset{def}{=} (repair1, \eta_1).Server_{standby} \\
Server_{standby} &\overset{def}{=} (repair2, \eta_2).Server_{on}
\end{aligned}$$

$$Queue_0 \underset{\{service, arrival\}}{\bowtie} Server_{on}$$

3.2 Excluded shared actions

In [8] Hillston and Thomas identified the limitations of their approach as being that they could not handle cases where the components were not reversible, but the resultant model was.

$$\begin{aligned}
P_0 &\overset{def}{=} (in_P, r).P_1 + (a, r).P'_1 \\
P_i &\overset{def}{=} (in_P, r).P_{i+1} + (out_P, s).P_{i-1} \\
&\quad 1 \leq i \leq N-1 \\
P_N &\overset{def}{=} (out_P, s).P_{N-1} \\
P'_i &\overset{def}{=} (a, r).P'_{i+1} \\
&\quad 1 \leq i \leq M-1 \\
P'_M &\overset{def}{=} (a, r).P_0
\end{aligned}$$

$$\begin{aligned}
Q_0 &\overset{def}{=} (in_Q, r).Q_1 + (b, r).Q'_1 \\
Q_i &\overset{def}{=} (in_Q, r).Q_{i+1} + (out_Q, s).Q_{i-1} \\
&\quad 1 \leq i \leq N-1 \\
Q_N &\overset{def}{=} (out_Q, s).Q_{N-1} \\
Q'_i &\overset{def}{=} (b, r).Q'_{i+1} \\
&\quad 1 \leq i \leq M-1 \\
Q'_M &\overset{def}{=} (b, r).Q_0
\end{aligned}$$

$$Sys \overset{def}{=} P \underset{\{a,b\}}{\bowtie} Q$$

In this model it might appear that the P component controls the Q component over the action b and the Q component controls the P component over the action a. However, $b \notin \boldsymbol{\mathcal{A}}(P_0)$ and $a \notin \boldsymbol{\mathcal{A}}(Q_0)$, hence both these shared actions are permanently blocked. Therefore it is clear that no control is exerted between these two components, both are behaviourally independent and the model has a trivial product form solution when the excluded behaviours are removed from the components (the rates of the actions a and b are always zero). However, this is not universally true for excluded behaviours. For example, consider the case where P'_M is redefined as follows.

$$P'_M \overset{def}{=} (b, r).P_0$$

Now $b \in \boldsymbol{\mathcal{A}}(P_0)$, but in this model b will never happen since action a is required to reach P'_M and action a is always blocked. Both components are still behaviourally independent by definition 3.1, however the tests specified in Section 3.2 will not identify this. In general it is not a simple matter to know that an action will not occur unless the states of the underlying CTMC are explored, therefore cases of excluded actions such as this are, in general, extremely problematical when working at the component level.

3.3 Extended example

In this simple example there are a pair of resources. The only stipulation is that at least one must be online at any time. Thus, any resource may choose to go offline only if another remains online. The correctness is held by the fact that in order to make the transition from $xOnline$ to $xOffline$, each resource must have the (passive) cooperation of its partner. Once offline however, the resources are incapable of communicating, and so the resource which is online must remain so.

$$\begin{aligned}
AOnline &\overset{def}{=} (Aoff, r_2).AOffline + \\
&\quad (Boff, \top).AOnline \\
AOffline &\overset{def}{=} (on, r_4).AOnline \\
BOnline &\overset{def}{=} (Boff, r_2).BOffline \\
&\quad +(Aoff, \top).BOnline \\
BOffline &\overset{def}{=} (on, r_6).BOnline
\end{aligned}$$

$$AOnline \underset{\{Aoff,Boff\}}{\bowtie} BOnline$$

Thus the model has the excluded state of $AOffline|BOffline$, and a trivial product form over the remaining states, given by,

$$\pi_{(AY,BZ)} = \frac{1}{X}\pi_{AY}.\pi_{BZ}$$

where $Y, Z = \{Online, Offline\}$ and $X = \pi_{AOnline}.\pi_{BOnline} + \pi_{AOnline}.\pi_{BOffline} + \pi_{AOffline}.\pi_{BOnline}$. The model can be made slightly more interesting if resource perform some computation. There are a number of possibilities in this regard.

– *Both resources compute together at all times.*

$$\begin{aligned}
AOnline &\overset{def}{=} (compute, r_7).ABusy \\
&\quad +(Aoff, r_2).AOffline \\
&\quad +(Boff, \top).AOnline \\
ABusy &\overset{def}{=} (complete, r_8).AOnline \\
BOnline &\overset{def}{=} (compute, r_7).BBusy \\
&\quad +(Boff, r_2).BOffline \\
&\quad +(Aoff, \top).AOnline \\
BBusy &\overset{def}{=} (complete, r_8).BOnline
\end{aligned}$$

$$AOnline \underset{\substack{\{compute \\ Aoff,Boff\}}}{\bowtie} BOnline$$

– *One of the resources is busy, the other must be online.*

$$\begin{aligned}
AOnline &\overset{def}{=} (compute, r_7).ABusy \\
&\quad +(compute, \top).AOnline \\
&\quad +(Aoff, r_2).AOffline \\
&\quad +(Boff, \top).AOnline \\
ABusy &\overset{def}{=} (complete, r_8).AOnline \\
BOnline &\overset{def}{=} (compute, r_7).BBusy \\
&\quad +(compute, \top).BOnline \\
&\quad +(Boff, r_2).BOffline \\
&\quad +(Aoff, \top).BOnline \\
BBusy &\overset{def}{=} (complete, r_8).BOnline
\end{aligned}$$

$$AOnline \underset{\substack{\{compute\\ Aoff, Boff\}}}{\bowtie} BOnline$$

This case only has a product form if $r_7 = r_9$.

4 Conclusions and Further Work

In this paper a discussion of the notions of behavioural independence and control has been presented in relation to product form solution. It is probable that many additional classes of product form solution will have subclasses that can be defined using behavioural independence and control, although this remains to be proved. The simple product form here can be extended by considering parts of components as behaviourally independent and parts which exert control. This ongoing work aims to build simple relations for models which may be more generally characterised by the definitions in [7]. By exploring these subclasses of solutions it will be possible to gain greater understanding of the links between different product form solutions (and non-product form solutions) and where they may overlap. In addition, practical approaches to the identification of these properties allow automation of model decomposition and hence greater support for the performance modeller.

References

1. Hillston J. Exploiting Structure in Solution: Decomposing Composed Models, in: *FMPA Lecture Notes*, Springer-Verlag, 2001.
2. Thomas N. and Bradley J. and Thornley D. Approximate solution of PEPA models using component substitution *IEE Proceedings - Computers and Digital Techniques*, **150**(2), pp. 67-74, 2003.

3. Thomas N. and Zhao Y. Mean value analysis for a class of PEPA models *The Computer Journal*, **54**(5), 643-652, 2011.
4. J. Hillston, Fluid flow approximation of PEPA models, in: *Proceedings of QEST'05*, pp. 33-43, IEEE Computer Society, 2005.
5. Zhao Y. and Thomas N. Efficient solutions of a PEPA model of a key distribution centre *Performance Evaluation*, **67**(8), pp. 740-756, 2010.
6. Harrison P.G. and Hillston J. Exploiting Quasi-reversible Structures in Markovian Process Algebra Models, *The Computer Journal*, **38**(7), pp. 510–520, 1995.
7. Hillston J. and Thomas N. Product Form Solution for a Class of PEPA Models, *Performance Evaluation*, **35**(3-4), pp. 171-192, June 1999.
8. Hillston J. and Thomas N. A Syntactic Analysis of Reversible PEPA Models, in: *Proceedings of the Sixth International Workshop on Process Algebra and Performance Modelling*, 1998.
9. Harrison, P.G., Turning back time in Markovian process algebra, *Theoretical Computer Science*, **290**(3): pp. 1947-1986, 2003.
10. Harrison PG. and Thomas N. Product-form solution in PEPA via the reversed process in: *Network performance engineering*, pp. 343-356, 2011
11. Harrison PG. and Thomas N. Semi-Product-Form Solution for PEPA Models with Functional Rates in: *Analytical and Stochastic Modeling Techniques and Applications*, pp. 416-430, 2013.
12. Hillston J. *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.
13. Thomas N. Exploiting behavioural independence and control in Markovian process algebra, in: *Proceedings of the 1st Workshop on Process Algebra with Stochastic Timed Activities*, University of Edinburgh, 2002.
14. Harrison PG. and Thomas N. State-dependent rates and semi-product-form via the reversed process in: *Computer Performance Engineering*, pp. 207-218, 2010.
15. Thomas N. Using ODEs from PEPA models to derive asymptotic solutions for a class of closed queueing networks Technical Report CS-TR-1129, School of Computing Science, Newcastle University, 2008